

REMARKS

This is a full and timely response to the outstanding final Office Action mailed October 23, 2003. Reconsideration and allowance of the application and presently pending claims 21-43 are respectfully requested.

1. Present Status of Patent Application

Upon entry of the amendments in this response, claims 21-43 remain pending in the present application. More specifically, claims 1-20 are canceled and claims 21-43 are newly added. These amendments are specifically described hereinafter. It is believed that the foregoing amendments add no new matter to the present application.

2. Request for Continued Examination

In accordance with 37 U.S.C. 1.114, a Request For Continued Examination is filed concurrently with this Response To The Final Office Action so that the Office Action mailed October 26, 2000 (Paper No. 8) is effectively made non-final.

3. Response to Rejection of Claims Under 35 U.S.C. §102 and §103

In the Office Action, claims 1, 4, 6, 9, 11, 14, 16 and 19 stand rejected under 35 U.S.C. §102(b) as allegedly being unpatentable by *Agarwal* (U.S. Patent 5,966,541), hereinafter *Agarwal*. In the Office Action, claims 2-3, 5, 7-8, 10, 12-13, 15, 17-18 and 20 stand rejected under 35 U.S.C. §103(a) as allegedly being unpatentable over *Agarwal* in view of *Brunmeier* (U.S. Patent 5,511,164), hereinafter *Brunmeier*.

Claims 1-20 are canceled without prejudice, waiver, or disclaimer, and therefore, the rejection to these claims are rendered moot. Applicant takes this action merely to reduce the number of disputed issues and to facilitate early allowance and issuance of other claims in the present application. Applicant reserves the right to pursue the subject matter of these canceled claims in a continuing application, if Applicant so chooses, and does not intend to dedicate any of the canceled subject matter to the public.

Applicant thanks the Examiner for his thorough Office Action wherein the features of *Agarwal* are described in detail. This detailed description of *Agarwal* has helped the Applicant appreciate the Examiner's interpretation of *Agarwal*.

4. Characterization of Agarwal

Applicant has thoroughly analyzed *Agarwal* and believe that *Agarwal* fails to disclose, for example, the features of “testing an assertion in the generated function code, such that when the assertion is true, the generated function code performs the steps of: generating a hidden failure code; *returning* to the generated program code; *and returning the hidden failure code* to the generated program code” as recited in claim 21 (emphasis added).

Generally, *Agarwal* discloses “a method for repairing or testing for many type mismatch problems in programs works by transforming a binary representation of the program into a new binary in which the problem is fixed or identified. The fix or identification is *implemented by converting code* that operates on variables that can suffer a mismatch into code that correctly accounts for or tests for the mismatch. Static or dynamic correlation methods, and/or control and data flow graphs are *used to track certain values*, to *determine where to install patches* and how to adjust branch, jump and procedure call references after patch installation has shifted the target references” (emphasis added, Abstract). Here, at most, the *Agarwal* Abstract discloses that code is converted, certain values are tracked by the control and data flow graphs, and a determination is made on “where” to install patches.

Agarwal discloses an embodiment that “performs assertion checking wherein a faulty or seemingly working binary is instrumented and potential errors are *flagged*. For example, the instrumentation can look for date-holding registers or memory locations in which the third and fourth digits are zeros and *flag a potential error condition* so that *a user can look* at the code and verify whether it is a real problem.” (Emphasis added, Col. 4, lines 57-63.) Applicant observes that this *Agarwal* flag is *limited to providing information to the user*.

Agarwal then discloses that in another embodiment, “running instrumented code can provide a quantitative assessment of the coverage provided by the tests. This method can also *flag* paths that did not get tested. Coverage information can be provided on a function by function basis.” (Emphasis added, Col. 5, lines 2-5.) Applicant observes that this *Agarwal* flag is limited to flagging paths that did not get tested. *Agarwal* provides additional detail regarding this flag type by disclosing that “many users outsource their code to vendors who manually fix some problem. The vendors test the fixed code with a suite of tests, and may also perform regression tests to check that the fixes did not break other parts of the code. Running instrumented

code can provide a quantitative assessment of the coverage provided by the tests. This method can also flag paths that are not tested.” (Col. 11, lines 5-11.)

Agarwal further discloses that “while binary rewriting techniques have been utilized for cross-platform execution, the present invention uses binary rewriting for code testing, protection, *error flagging*, and remediation.” (Emphasis added, Col. 5, lines 64-66.) Applicant observes that *Agarwal* does not disclose any functionality of the “error flag” above, and accordingly, additional information from the *Agarwal* specification is required to properly interpret the “error flag” above.

Agarwal discloses flagging error conditions by stating that “as the instrumented code is continuously executed, the new instruction is continuously on the look-out for an error condition (the value in date.a is less than 1900), which will be flagged should it occur. Note that although the sample program segments above are shown in pseudo-code, the present invention uses the binary representations. Of course, other kinds of tests are possible. For example, the patch could check that the two significant digits in date.a are either “19” or “20” and flag any case in which this is not true. A patch can test for virtually any condition. A user can also be queried to provide valid ranges for values that can then be used in self-testing.” (Emphasis added, Col. 12, line 66 through Col. 13, line 10.) Applicant observes that this *Agarwal* flag is *limited to flagging error conditions*. *Agarwal* does not disclose precisely what is done with the flagged error condition.

Agarwal discloses yet another type of flagging, at Col. 11, lines 31-49, repeated below for the convenience of the Examiner:

The present embodiment creates an instrumented binary 701 (FIG. 11) from an original binary 700 by inserting a patch PATCH100-PATCH103 at the beginning of each BLOCK100-BLOCK103 respectively (step 109 of FIG. 10). In the patched binary, some memory 703 is allocated for test coverage such that some memory, perhaps a single bit, is associated with each program block. The memory is first initialized to all zeroes as shown. It can be seen that when a block of code such as BLOCK100 is about to be executed, the associated patch PATCH100 executes first. PATCH100 simply sets to 1 the bit in memory associated with BLOCK100. This is shown symbolically with arrow 705. Similarly, when BLOCK101 is about to execute, PATCH101 executes first, setting the bit associated with BLOCK101 as shown by arrow 707. Ultimately, any block of code that has executed will be flagged by having its associated bit set to 1, while any block that has not executed will have its associated bit equal to 0.

Here, *Agarwal* is limited to disclosing flags that identify blocks of code that are executed and blocks of code that are not executed.

Finally, *Agarwal* discloses another type of flag associated with coloring by disclosing another embodiment that “performs test path identification. In conjunction with user-supplied information such as date entry points, this approach can also analyze the code in order to **flag (color)** the parts of the program that might get corrupted with, in the case of date remediation, a date variable.” (Emphasis added, Col. 5, line 34-39.) *Agarwal* discloses the general nature of coloring in the Background section, at Col. 2, lines 34-48, repeated below for the convenience of the Examiner:

In either the windowing or expansion technique, the simplest methods require searching through all of the source code, or using some dynamic method to track corrupted values. One approach to reducing the search space uses program coloring and works as follows. A user might be required to submit the names of all variables that might contain a date. A program flow analysis at the source program level then identifies all regions in the program where data from the named variables might flow and thereby have an effect. The regions of the program where the named variables might have an effect are designated as “colored” regions. The programmer need only look at the colored regions to implement the fixes.

Agarwal describes how “coloring” is employed in flow graph coloring step 83.

Agarwal discloses that:

Referring back to FIG. 1B with the aid of the data flow graph, the next step 83 is to identify or “color” the instructions that potentially use dates or selected arguments. Starting with instructions identified as using dates (or specific arguments) or as being instructions that obtain a date through a program input, **data analysis is used to mark or color all the instructions that can be contaminated** with a date (or with the specific argument). FIG. 7 shows a colored graph 400 for the case where variable b is a date. In this graph, the hashed nodes N2, N3, N4, N5, and N8 correspond to the instructions that may have to be changed.

Again referring to FIG. 1B, the actual rewriting 85, 87, 89 of the binary now takes place. First, the patches are installed 85. **Each colored binary instruction is replaced** by a set of binary instructions that implement the correct logic. For example, the instruction I3, $c=a+b$, is replaced in a manner similar to that described earlier. (Emphasis added, Col. 9, line 66 through Col. 10, line 15.)

Applicant observes that this type of flag in *Agarwal* is limited to “coloring” blocks and/or lines of code. Coloring is further explained in *Agarwal* at Col. 12, lines 13-33 (emphasis added), repeated below for the convenience of the Examiner:

New and modified blocks from the modified binary are "colored" in step 159, and finally the *colored blocks are selected* (step 173). Control flow or data flow analysis can also be used to additionally *color lines of code* that are affected by the modified or new lines.

Argument remediation coverage testing is similar to modified code coverage testing. In this case, however, the bug fix is related to arguments passed into the program, e.g., entered by a user, such as age. Where the original binary may have accepted age in years, the modified binary may be instrumented to accept an age in months. In this case, path 161 is taken. A data flow graph is generated (step 163) and colored following a specific argument (step 165). Finally, the colored blocks are selected (step 173).

Date remediation coverage test is simply a special case of argument remediation coverage testing, where the argument to be followed is a date. Path 167 is taken. Again, a data flow graph is generated (step 169). Now, the graph is colored using dates (step 171). Finally, the colored blocks are selected (step 173).

Accordingly, Applicant believes that *Agarwal* fails to disclose, for example, the features of "testing an assertion in the generated function code, such that when the assertion is true, the generated function code performs the steps of: generating a hidden failure code; *returning* to the generated program code; and returning the hidden failure code to the generated program code" as recited in claim 21 (emphasis added). Nor does *Agarwal* disclose, teach or suggest an alternative embodiment, "step of generating a hidden failure code further comprises generating a hidden failure flag by the generated function code" (as recited in claim 22).

5. Newly Added Claims 21-43

New claims 21-43 are based on subject matter that is explicit and/or inherent within the description of the specification and/or the drawings. Applicant submits that no new matter has been added in the new claims 21-43, and that new claims 21-43 are allowable over the cited prior art. Therefore, Applicant requests the Examiner to enter and allow the above new claims.

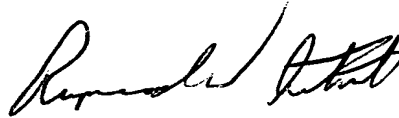
6. Amendments to the Specification

The amendments made to the specification are all based on subject matter that is explicit and/or inherent within the description of the specification and/or inherent within the drawings, and merely place the Application in better form for allowance. Applicant submits that no new matter has been added in the specification, and request the Examiner to enter the above amendments into this Application.

CONCLUSION

In light of the foregoing amendments and for at least the reasons set forth above, Applicant respectfully submits that all objections and/or rejections have been traversed, rendered moot, and/or accommodated, and that the now pending claims 21-43 are in condition for allowance. Favorable reconsideration and allowance of the present application and all pending claims are hereby courteously requested. If, in the opinion of the Examiner, a telephonic conference would expedite the examination of this matter, the Examiner is invited to call the undersigned agent at (770) 933-9500.

Respectfully submitted,



Raymond W. Armentrout
Reg. No. 45,866